rachit tayal lab2

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image_path = "/content/OIP.jpeg"
image = cv2.imread(image_path)
if image is None:
    raise FileNotFoundError(f"Image not found at {image_path}")

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for display
original_height, original_width = image.shape[:2]
print(f"Original Dimensions: Width={original_width}, Height={original_height}")

# Function to display images
def display_images(images, titles, cols=2):
    rows = (len(images) + cols - 1) // cols
    plt.figure(figsize=(10, 5 * rows))
    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(img, cmap="gray" if len(img.shape) == 2 else None)
        plt.title(title)
        plt.axis("off")
    plt.tight_layout()
    plt.show()

# 1.1 Image Resizing
linear_resized = cv2.resize(image_rgb, (300, 300), interpolation=cv2.INTER_LINEAR)
nearest_resized = cv2.resize(image_rgb, (230, 230), interpolation=cv2.INTER_NEAREST)
cubic_resized = cv2.resize(image_rgb, (200, 200), interpolation=cv2.INTER_CUBIC)

# 1.2 Image Blurring
box_blurred = cv2.blur(image_rgb, (35, 35))
gaussian_blurred = cv2.GaussianBlur(image_rgb, (25, 25), 0)
adaptive_blurred = cv2.medianBlur(image_rgb, 15)
```

```
# Display results
display_images(
    [
        image_rgb,
        linear_resized,
        nearest_resized,
        cubic_resized,
        box_blurred,
        gaussian_blurred,
        adaptive_blurred,
    ],
    [
        "Original Image",
        "Linear Resized (300x300)",
        "Nearest Neighbor Resized (230x230)",
        "Cubic Resized (200x200)",
        "Box Blurred",
        "Gaussian Blurred",
        "Adaptive (Median) Blurred",
    ]
)
```

```
Original Dimensions: Width=474, Height=315
```

Original Image

Linear Resized (300x300)

Nearest Neighbor Resized (230x230)

Cubic Resized (200x200)

Box Blurred

Gaussian Blurred

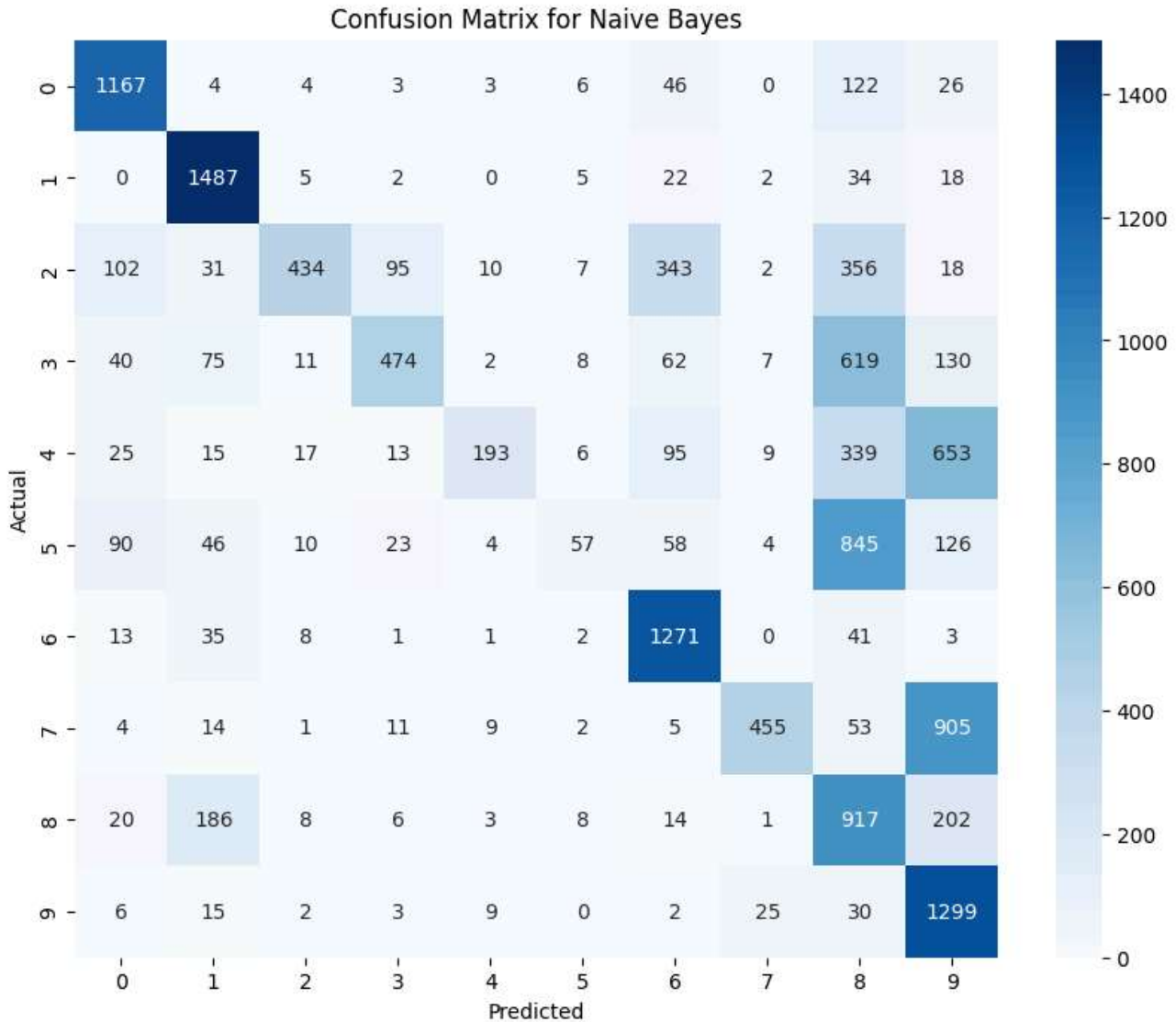
Adaptive (Median) Blurred

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    roc_curve,
    auc
)
import matplotlib.pyplot as plt
import seaborn as sns

# Load MNIST dataset
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist.data, mnist.target.astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Initialize algorithms
models = {
    "Naive Bayes": GaussianNB(),
    "Random Forest": RandomForestClassifier(random_state=42),
}

# Perform k-fold cross-validation and evaluation
results = {}
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for model_name, model in models.items():
    print(f"\nTraining {model_name}...")
```

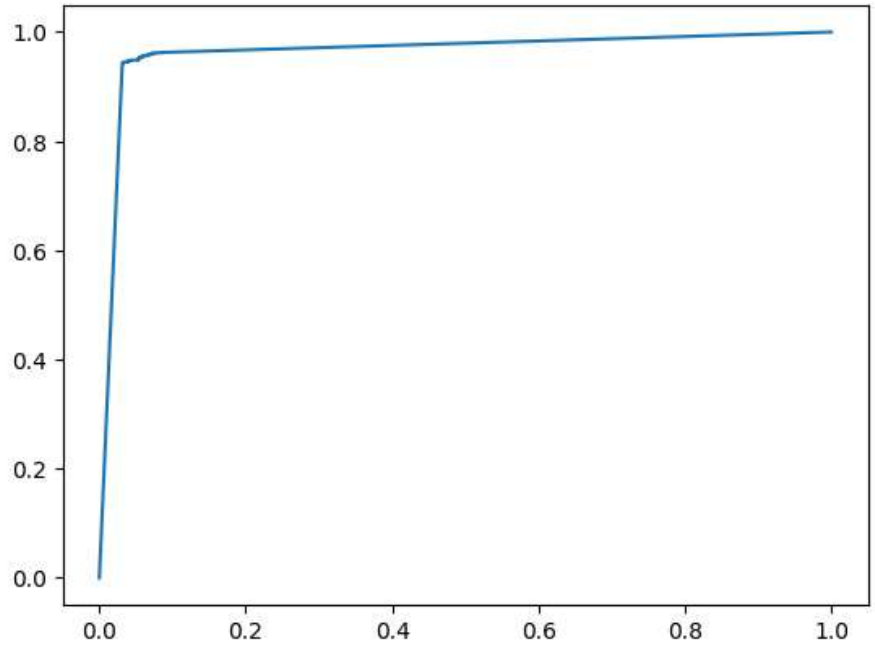
```
model_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='accuracy')

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

y_proba = model.predict_proba(X_test) if hasattr(model, "predict_proba") else None

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title(f"Confusion Matrix for {model_name}")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ROC and AUC (for binary classifiers; this is a workaround for digit 1 vs others)
if y_proba is not None and y_proba.shape[1] > 1:
    fpr, tpr, _ = roc_curve(y_test == 1, y_proba[:, 1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")
else:
    roc_auc = None

# Store results
results[model_name] = {
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1-Score": f1,
    "Confusion Matrix": conf_matrix,
    "AUC": roc_auc
```

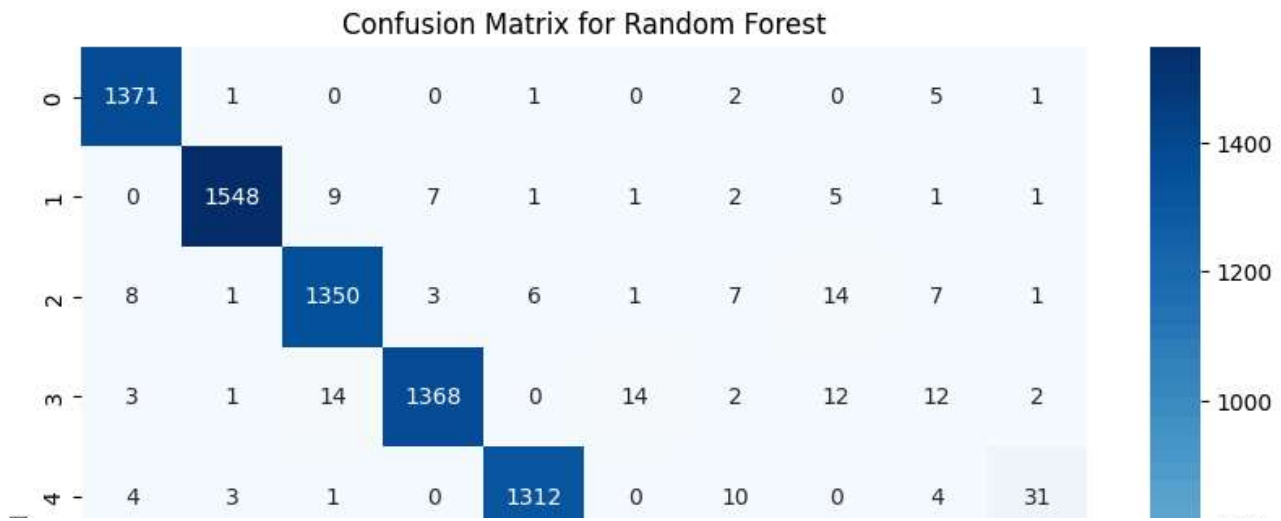
```
    }

# Plot ROC curves
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc='best')
plt.show()

# Print results
for model_name, metrics in results.items():
    print(f"\nResults for {model_name}:")
    for metric, value in metrics.items():
        if metric == "Confusion Matrix":
            print(f"{metric}:\n{value}")
        else:
            print(f"{metric}: {value:.4f}")
```

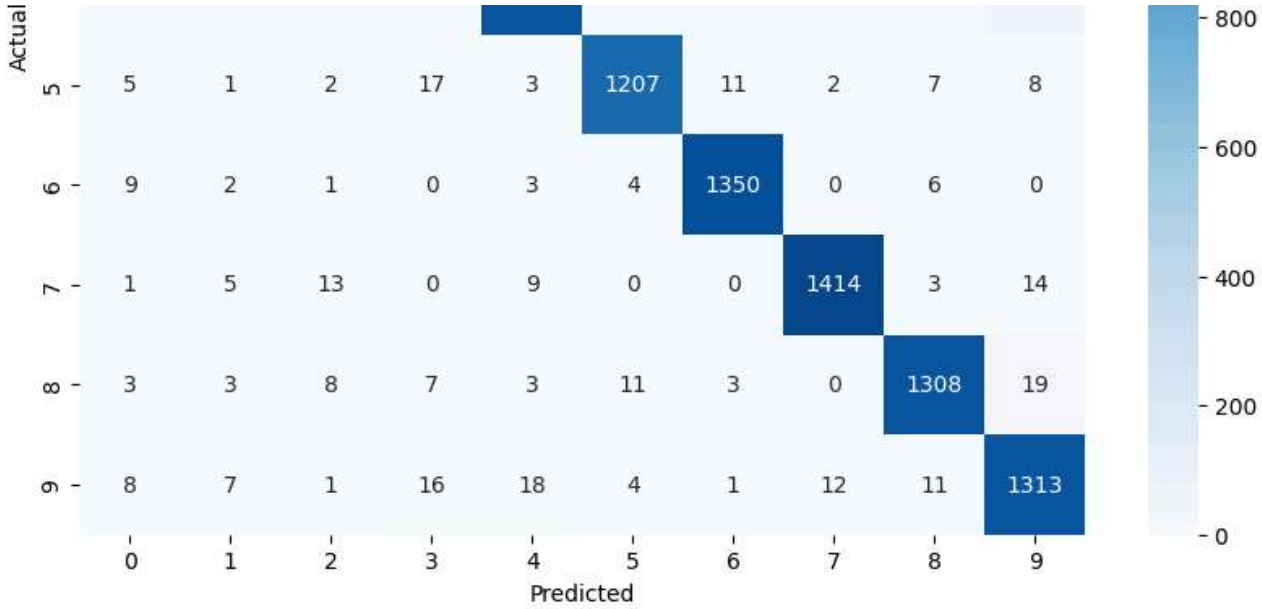
Training Naive Bayes...

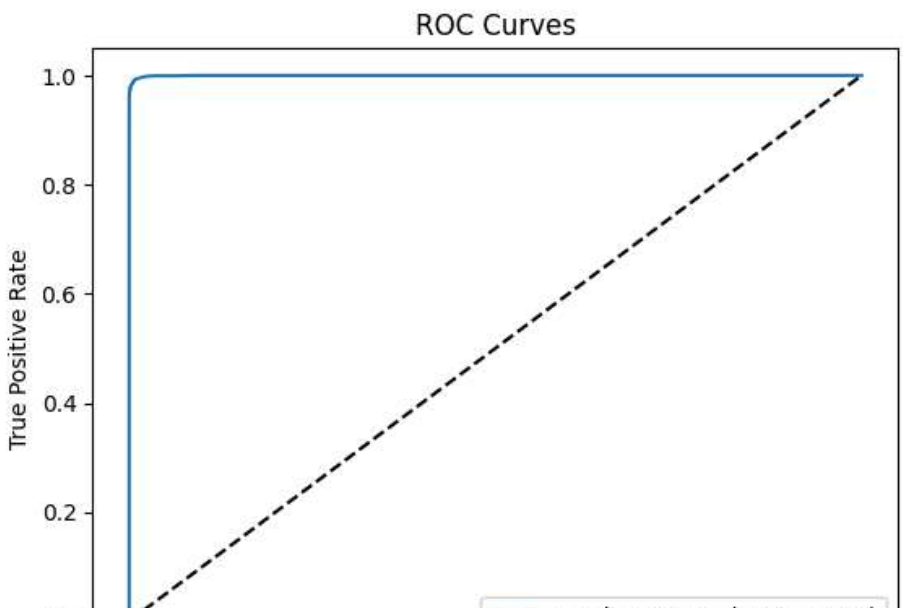


```
Training Random Forest...
```






Actual
5 5 1 2 17 3 1207 11 2 7 8
6 9 2 1 0 3 4 1350 0 6 0
7 1 5 13 0 9 0 0 1414 3 14
8 3 3 8 7 3 11 3 0 1308 19
9 8 7 1 16 18 4 1 12 11 1313
0 1 2 3 4 5 6 7 8 9
Predicted
800
600
400
200
0


ROC Curves
True Positive Rate
1.0
0.8
0.6
0.4
0.2